Programming Assignment 4 – CPU Scheduling Policies

Assigned: Tuesday, 4/11/2006, 11am
Due: Tuesday, 4/25/2006 before 4pm

In this assignment, you will write a program that simulates the CPU scheduling algorithms described in section 9.2 of the required textbook by William Stallings (2004) *Operating Systems: Internals and Design Principles (5th Edition)*: First-Come-First-Serve (FCFS), Shortest Process Next (SPN) and Shortest Remaining Time (SRT). ***Graduate students only (extra credit for the others):*** Implement Round-Robin (RR), too, using only one queue for interrupted and new processes (as in Figure 9.5, 2nd and 3rd tiers). This project contains three separate executables: a CPU scheduling simulator `schedsim`, a process arrival generator `schedgen`, and a time statistics program `schedstats` that produces consolidated data files used for performance plots.

1. Specifications of the CPU scheduling simulator `schedsim`

- ✓ `schedsim` must accept two command-line arguments: (a) an input filename where a sequence of process arrival & burst times is stored, and (b) the chosen algorithm. The three (***grads:*** four) possible values for (b) are the strings `fcfs`, `spn` and `srt` (***grads:*** and `rr`). ***Graduates:*** Your program must also accept a third argument: (c) the round-robin time quantum q (positive integer), if and only if the second argument is `rr`. Command line examples:

  ```
  >./schedsim schedtimes.dat fcfs
  >./schedsim schedtimes.dat rr 4
  ```

  If `schedsim` is run with the wrong arguments or no arguments, it should print out a few lines summarizing the usage instructions and exit.

- ✓ The format of the input file `schedtimes.dat` must be a simple ASCII list of pairs of integers separated by line breaks, for example:

  ```
  5 9
  7 3
  8 3
  9 11
  ```

  The first integer value represents the arrival time of a process and the second integer value represents the burst time (in milliseconds, with one space between the two values). The process number is implicitly given by the line number, starting at 1. Arrival times must be sorted in increasing order. The input file should contain no other symbols or text. It can be written by hand or generated by the `schedgen` program (see section 2. below).

- ✓ `schedsim` will first read all the process values from the input file and store them in a local array. Then, it will play back these events and output the resulting schedule chart in character mode format, along with the end times, turnaround times and waiting times for each completed process. For example, the time values corresponding to the SRT section of Figure 9.5 of the textbook must be displayed as follows:

```
1: runs 0-3 -> end=3, (arr=0), turn=3, (burst=3), wait=0
2: runs 3-4
3: runs 4-8 -> end=8, (arr=4), turn=4, (burst=4), wait=0
5: runs 8-10 -> end=10, (arr=8), turn=2, (burst=2), wait=0
2: runs 10-15 -> end=15, (arr=2), turn=13, (burst=6), wait=7
4: runs 15-20 -> end=20, (arr=6), turn=14, (burst=5), wait=9
```

This representation means that process 2 (called B in the figure) ran from time 3 to time 4, then ran again from time 10 to time 15 and completed at time 15, therefore its turnaround time is 15 (end) – 2 (arrival) = 13 and its waiting time is 13 (turnaround) – 6 (burst) = 7. The timing values must be displayed exactly as illustrated above, all on the same line as the last process burst, using the characters and symbols as shown. Do *not* align values using tab characters.

✓ After handling all the process events, `schedsim` should finally print the global average turnaround time, global average normalized turnaround time (average of turnaround times divided by burst times, i.e., average of the divisions, *not* division of the averages) and global average waiting time. Use this printout format with two decimals:

```
Average turnaround time = 7.20
Average normalized turnaround time = 1.59
Average waiting time = 3.20
```

2. Specifications of the process arrival generator `schedgen`

✓ `schedgen` must accept four command-line arguments in the following order: (a) the total number of processes N (positive integer), (b) the probability of arrival of a process p_a (real number between 0 and 1), (c) the probability of burst length of a process p_b (real number between 0 and 1), and (d) the name of the file that will be generated. Example:

```
>./schedgen 1000 0.5 0.2 schedtimes.dat
```

If `schedgen` is run with the wrong arguments or no arguments, it should print out a few lines summarizing the usage instructions and exit.

✓ `schedgen` will generate a list of N processes formatted as specified in Section 1 above. The time values are to be chosen randomly as follows: at each time step t_a = 0, 1, 2, ... if a real random number uniformly drawn in [0, 1) is less than p_a, it means that a new process arrived at t_a. In this case, the burst time of this process t_b is going to be determined by launching a second time step counter t_b = +1, +2, +3, ... and stopping whenever a second real random number uniformly drawn in [0, 1) happens to be less than p_b. At that point, and only then, a new line containing the pair t_a t_b is added to the generated file. Repeat until the file contains N lines.

3. Specifications of the CPU scheduling statistics program `schedstats`

✓ `schedstats` must accept only one command-line argument: (a) the input filename containing the process times. Example:

```
>./schedstats schedtimes.dat
```

If `schedstats` is run with the wrong arguments or no arguments, it should print out a few lines summarizing the usage instructions and exit.

- ✓ `schedstats` will loop over three (***grads:*** five) CPU scheduling methods: `fcfs`, `spn`, `srt` (***grads:*** and `rr` with $q = 1$ and $q = 4$) and for each method, it will calculate a histogram of average time values per burst length. This means that a separate average turnaround time, average normalized turnaround time and average waiting time will be calculated for each burst length $t_b = +1, +2, +3, ...$ by pooling together all processes having this burst time.

- ✓ while running, `schedstats` will only print out one user message per algorithm to indicating its current execution status:

```
processing fcfs...
processing spn...
...
```

- ✓ most importantly, `schedstats` must generate three results files containing all the calculated average times. These results files must be named: `schedturns.dat` (average turnaround times), `schednturns.dat` (average normalized turnaround times) and `schedwaits.dat` (average waiting times), and must be formatted in the following way: Each file should contain exactly four (***grads:*** six) lines of B space-separated numbers, where B is the number of different burst times. The first line must be the ordered sequence of different burst times extracted from `schedtimes.dat` (*likely* to be `1 2 3 4 ...`, with possible gaps), the other three (***grads:*** five) lines must be the sequences of average time values for each burst time. These sequences must be in the following order: `fcfs`, `spn`, `srt` (***grads:*** and `rr` with $q = 1$ and $q = 4$).

- ✓ Therefore, the implementation of both `schedsim` and `schedstats` must rely on a common utility simulation engine (entry-point function) with the difference that `schedsim` is going to run the simulation once and in verbose mode (displaying all the time steps and the global average values; see Section 1), whereas `schedstats` is going to run the simulation repeatedly and silently, and calculate per-burst averages, not global averages. In accordance with this design, please keep the simulation engine in a source file distinct from the `schedsim` and `schedstats` capstone codes.

Specifications of the plots

- ✓ Once `schedturns.dat`, `schednturns.dat` and `schedwaits.dat` have been generated you must use the graphing tool of your choice to create three plots, each plot containing the three (***grads:*** five) scheduling algorithms, similarly to Figures 9.14 and 9.15 of the textbook. Each plot file should be a JPEG (or PNG) image respectively called `schedturns.jpg`, `schednturns.jpg` and `schedwaits.jpg` (or `.png`).

- ✓ A benchmark input file will be sent to you later. Along with your source and documentation files, you will have to submit the stats files and plot image files resulting from the execution of `schedstats` against this input file.

Project Requirements, Submission & Required Documentation

- ✓ Follow all the requirement and submission guidelines *exactly* as prescribed in Stallings' book on pages 154 to 156, replacing the name of the executable `myshell` with `schedsim`, `schedgen` and `schedstats`, and naturally any mention of "shell" and shell features with these programs' features. I remind you that Stallings' guidelines include, among other things: appropriately structuring and properly commenting the code (3.), not including object code or executables in your submission (5.), naming the makefile exactly `makefile` (6.), naming the readme file exactly `readme`, and writing a reasonably detailed UNIX-like readme documentation (2.). Your documentation should describe the problem's background, the programs' usage and parameters, and a sample of a typical outcome.

- ✓ Your programs *must* compile without warnings and run properly under Gentoo Linux, the system used in the ECC lab (check their Web site for hours). You may develop and test your programs on your own UNIX machine (you can also produce the stats file and plots from your machine), but it is your responsibility to make sure that they also work properly on the Gentoo installation of ECC, under the default ECC Lab shell. There will be a penalty if they don't.

- ✓ Place all necessary files and documents in a compressed tarball using tar and gzip (`.tar.gz` or `.tgz`) or, alternatively, a zip file (`.zip`) (please do not use the `bzip2` format). Email this file to the instructor doursat@unr.edu before the deadline above. Do not turn in printouts in any form. Late assignments will be marked down according to the late policy published on the course Web page.

- ✓ Do not show, exchange or copy code with anyone, and do not look for a solution on the Web! Plagiarism *will* be detected and severely penalized. This must remain an individual effort.

- ✓ Please keep clean code organization, layout and coding conventions, as these will also be an important part of the grade.

Thank you and good luck!